제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

strace를 이용한 버퍼오버플로우 탐지에 관한 연구

김희승*, 한영주**, 정태명*
*성균관대학교 컴퓨터공학과
**성균관대학교 전기전자 및 컴퓨터 공학과
e-mail : {hskim*, yjhan**}@imtl.skku.ac.kr, and
tmchung@ece.skku.ac.kr*

A Study on Attack Detection of Buffer Overflow using strace

Hee-Seung Kim*, Young-ju Han** and Tai-Myoung Chung*
*Dept. of Computer Engineering, SungKyunkwan University
**Dept. of Electrical and Computer Engineering, SungKyunkwan University

요 약

버퍼오버플로우 취약점은 컴퓨터 및 네트워크가 발전한 이래로 시스템 보안을 위협하는 가장 큰 문제점이었고, 현재 인터넷 웜과 결합된 형태로 발전하면서 그 문제점은 더욱 심각해 감에 따라 근본적으로 버퍼오버플로우를 방어할 수 있는 방법이 시급하다. 그러나 아직까지 효과적으로 버퍼오버플로우 공격을 탐지할 수 있는 방법이 제시되지 못하고 있다. 본 논문에서는 디버깅 프로그램인 strace를 이용하여 버퍼오버플로우 공격을 효과적으로 탐지할 수 있는 방법을 기술한다. strace를 이용한 버퍼오버플로우 탐지 방법은 호스트 기반의 침입 탐지 시스템에 효과적으로 적용될 수 있을 것이다.

1. 서론

해커 잡지인 Phrack 49호에 "Smashing the Stack for Fun and Profit"라는 기사로 일반인들에게 버퍼오버플로우(Buffer Overflow)가 소개되면서 많은 해커들이 이를 이용하여 서버를 공격했다[1]. 그간 대부분의 서버는 버퍼오버플로우 취약점을 가지고 있는 많은 응용에 의해 무방비 상태로 노출되어 있었다. 또한 최근에 버퍼오버플로우가 인터넷 웜과 결합된 형태로 발전하면서 그 피해가 심각한 상태에 달하고 있다[2]. 지금까지 canary 방식과 포인터 무결성 방식 등 몇몇 버퍼오버플로우 방어 대책이 연구되었다[3]. 그러나 canary 방식은 프로그램 작성 단계에서 canary 변수를 삽입하는 방법이기 때문에 기존의 프로그램에 존재하는 버퍼오버플로우의 가능성은 제거할 수 없으며, 포인터 무결성 방식은 리턴어드레스를 저장하고 비교하는 과정이 프로그램의 성능에 너무 많은 영향을 미친다는 단점이 있어 두 가지 방법 모두 한계를 드러내고 있다.

본 논문에서는 디버깅 프로그램인 strace를 이용하여 복잡한 과정 없이 탐지 역할을 하는 데몬 하나로 간단히 버퍼오버플로우를 탐지할 수 있는 방법을 제시하고자 한다. 2장에서는 버퍼오버플로우의 개념과 기존의 버퍼오버플로우의 방어 기법에 대해서 기술하고, 3장에서는 본

논문에서 제안하는 strace를 이용한 버퍼오버플로우 탐지 원리 및 방법에 대하여 논할 것이며, 마지막으로 4장에서는 결론과 향후 방향에 대해서 언급한다.

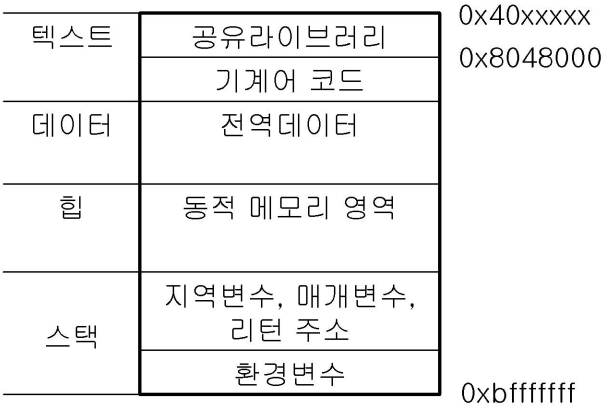
2. 관련연구

2.1. 버퍼오버플로우

버퍼오버플로우는 이름 그대로 프로그램 개발자가 선언해 놓은 버퍼를 의도적으로 넘치게 함으로써, 프로그램이 오동작하게 만드는 해킹 기법이다[1,4]. 버퍼오버플로우를 알기 위해서는 버퍼가 위치하고 있는 메모리에 대한 명확한 지식이 필요하다.

2.1.1. 메모리의 구조

리눅스에서는 바이너리 실행파일의 형식으로 ELF(Executable and Linking Format)형식을 사용한다. 리눅스 커널에서 ELF형식의 바이너리가 메모리에 로드되는 방식은 binfmt_elf.c에 들어있다. binfmt_elf.c에 보면 메모리의 구조는 대략적으로 [그림 1]과 같이 네 가지의 영역으로 구분된다.

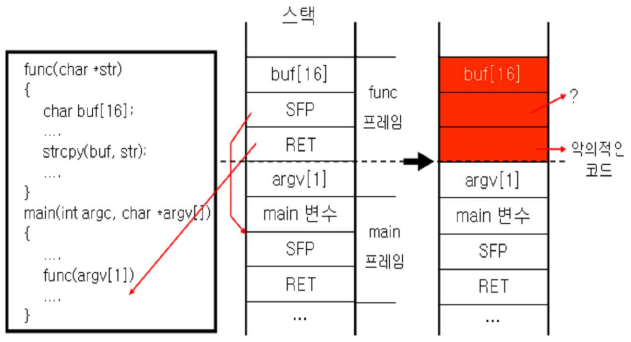


[그림 1] 메모리 구조

우선 텍스트(Text) 영역은 프로그램의 어셈블리 코드(assembly code)가 저장된다. 텍스트 영역은 공유라이브러리와 기계어 코드가 저장된다. 데이터 영역은 전역(global) 데이터들이 저장된다. 힙(Heap) 영역은 동적 변수(dynamic variables)가 사용하는 영역으로 C 언어에서 malloc()과 같은 시스템 호출을 통해서 할당되는 메모리는 힙 영역에 할당된다. 스택(Stack) 영역은 지역변수(local variables)가 사용하는 영역이다. 스택은 사용자 주소공간의 최상위 부분에 위치하며, LIFO(Last In, First Out)모델에 따라 동작한다. 함수가 호출될 때마다 지역 변수와 함수의 인자, 리턴어드레스 등을 위한 새로운 프레임이 스택에 생성되는데 CPU는 값을 집어넣는 push 연산과 값을 꺼내는 pop 연산을 통하여 스택을 제어한다. 메모리 구조에서 버퍼오버플로우의 대상이 되는 부분이 바로 이 스택 영역이다.

2.1.2. 버퍼오버플로우의 원리

현재의 C 언어에서는 일정한 양의 버퍼를 설정하고 그 버퍼에 채워질 값을 받을 경우, 그 값의 크기를 제한하지 않는다. 즉, 컴파일러는 일정한 양의 값을 기대하여 메모리를 잡아 놓지만, 사용자의 입력 값은 그 양과 무관하게 입력을 할 수 있다. 그 결과 버퍼가 넘침으로써 버퍼 뒤에 리턴어드레스를 저장하는 RET 영역을 침범하게 되고 프로그램의 흐름을 비정상적으로 만들게 된다. 이렇게 버퍼오버플로우의 가능성이 있는 프로그램과 버퍼오버플로우의 원리는 [그림 2]와 같다.



[그림 2] 버퍼오버플로우의 원리

[그림 2]의 프로그램에서 오버플로우의 대상이 되는 버퍼는 buf[16]이다. 그림과 같이 buf

공간은 스택 영역에 위치하게 되고, 이 공간 뒤에는 다시 main 함수로 복귀하기 위한 RET 영역이 존재한다. 버퍼가 넘치게 되면 RET 영역이 훼손되게 되어 엉뚱한 곳으로 리턴하게 된다. 해커는 이 위치에 쉘코드를 두어 새로운 쉘을 실행하게 한다. 쉘코드란 "/bin/sh"을 실행시키기 위한 프로그램 코드를 바이너리 형태로 변환한 것으로 [그림 2]의 프로그램에 관리자의 권한으로 setuid 비트가 설정되어 있다면, 실행도중 쉘코드에 의해서 관리자의 쉘을 얻을 수 있다[4].

2.2. 기존의 버퍼오버플로우 방어 기법

2.2.1. canary 방식

canary 방식은 버퍼오버플로우가 일어날 가능성이 있는 변수 앞에 특정 값으로 초기화된 canary 변수를 선언한 후 canary 변수 값의 변경 여부를 검사하므로써 버퍼오버플로우를 탐지하는 방식이다[3]. [그림 2]에서 버퍼오버플로우가 일어날 가능성이 있는 buf 변수 앞에 i라는 canary 변수를 선언하고, 예를 들어 i에 0x1234567라는 값을 넣어둔다. 버퍼가 오버플로우가 되었을 때, 오버플로우 된 데이터들이 i 변수의 영역을 침범하여 기존의 값을 변하게 되기 때문에 이후에 i값이 이전의 값과 동일한지 검사함으로서 버퍼오버플로우를 탐지할 수 있는 것이다. 하지만 이 방식은 해커가 i값을 쉽게 알 수 있어, i값도 조작하여 다시 기존의 값으로 바꿀 수 있다. 또한 프로그래밍 단계에서의 방어 기법이기 때문에 기존의 프로그램들에는 적용되지 못하는 단점이 있다.

2.2.2. 포인터 무결성 방식

포인터 무결성 방식은 버퍼오버플로우를 발생시키는 RET영역의 값을 안전하게 유지하므로써 근본적으로 버퍼오버플로우를 차단하고자 하는 방식이다[3]. 즉, RET 영역의 주소 값을 안전한 다른 곳에 복사해 두고 리턴을 하게 될 때 복사해둔 값의 주소를 참조하는 방식이다. 실제로 스택에 있는 RET 영역의 값이 버퍼오버플로우에 의해 변경되었을지라도 안전한 곳에 저장된 것을 참조하기 때문에 버퍼오버플로우 공격에 안전하게 된다. 하지만 이 방식은 프로그램의 성능을 급격히 저하시키므로 현실적이지 못하고 커널 수정이 필요하기 때문에 많이 이용되지 못하는 방식이다.

3. strace를 이용한 버퍼오버플로우의 탐지

3.1. strace

strace는 인자로 받는 프로그램에서 수행되는 모든 시스템 호출을 출력하며 어디에서 프로그램이 잘못 되었는지에 대한 정보를 출력하는 디버깅 프로그램 중에 하나이다[5]. strace는 ptrace 시스템 콜 함수를 이용하여 프로그램에 부착(attach)되어 해당 프로그램의 여러 정보를 가져온다. 이러한 정보는 프로그램의 동작과정을 설명하고 있기 때문에 해커에 의해 악의적으로 수정되었는지도 확인할 수 있다.

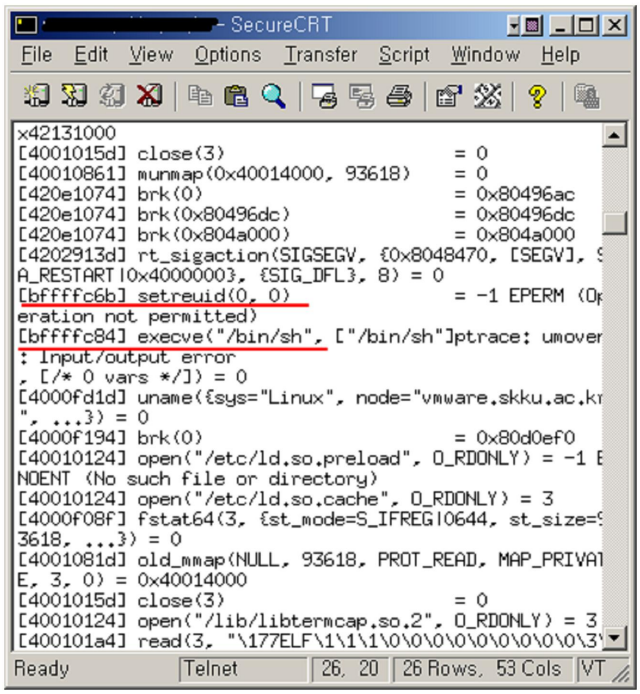
3.2. 탐지 원리

버퍼오버플로우에서 항상 사용되는 쉘코드는 항상 스택 영역에 저장되게 된다. 스택 영역의 메모리 주소는 0xbffffaa4와 같은 주소 영역이 된다. 이에 반해 프로그램의 실행 코드가 들어가는 텍스트 영역은 0x8048418과 같은 주소 영역이나 공유라이브러리가 위치하는 0x4001015d과 같은 주소 영역이 된다. 따라서 일반적인 프로그램에서는 실행되고 있는 명령어(instruction)의 주소를 추적해 보면 항상 텍스트 영역의 주소이지만, 버퍼오버플로우를 일으켜 쉘코드를 실행하고자 하면 명령어의 주소가 텍스트 영역에서 잠시 쉘코드의 실행을 위해 주소 영역이 스택 영역으로 바뀌게 된다. 이 특징을 이용하여 strace를 이용하여 사용자가 실행하는 프로세스를 감시하여 명령어의 주소가 잠시 스택

영역으로 바뀌게 되면 버퍼오버플로우가 발생했음을 알 수 있다. strace를 이용한 방식을 이용하면 기존의 프로그램에도 적용 가능하여 canary 방식의 단점을 보완할 수 있고, 간단하게 strace를 이용하여 프로세스를 감시하는 데몬 하나만을 만들면 되기 때문에 커널을 수정해야 하는 포인터 무결성 방식의 단점을 보완할 수 있다.

3.3 탐지 방법

[그림 2]와 같은 프로그램이 공격당할 때, 공격 프로그램의 명령어 주소를 볼 수 있도록 strace에 -i 옵션을 이용하여 실행하면 [그림 3]과 같은 결과를 볼 수 있다.



[그림 3] strace를 이용한 프로세스 추적

[그림 3]에서 밑줄을 그은 부분의 주소를 보면 0xbffffc84와 같이 스택 영역의 주소인 것을 알 수 있고, "/bin/sh"이 실행되고 있음을 보아 쉘코드가 실행 중이라는 것을 추측할 수 있다.

4. 결론 및 향후 방향

본 논문에서는 버퍼오버플로우를 효과적으로 탐지해낼 수 있는 방법을 제시하였다. 제시한 방법은 기존의 버퍼오버플로우의 방어 방식의 한계를 극복하고 보다 쉽고 간편한 방법으로 이루어진다. 따라서 호스트 기반의 침입탐지시스템에서 버퍼오버플로우 공격을 쉽게 탐지해낼 수 있을 것으로 예상된다.

하지만 우리가 제시한 방법은 사용자가 실행하는 모든 프로세스를 감시해야 하기 때문에 시스템의 자원을 많이 소모할 수 있다. 따라서 앞으로 어떠한 프로세스를 추적할 것인가를 결정하는 정책에 대한 연구 및 strace에 대한 연구를 더 진행하여 버퍼오버플로우 탐지 뿐 아니라 방어할 수 있는 방법을 연구할 것이다.

참고문헌

[1] Aleph One, "Smashing The Stack For Fun And Profit", Phrack 49-14.

[2] 전완근, 류성철, 김승철, "MS-SQL 서버 웜-슬래머(Slammer) 공격 테스트 및 사고대응", CERTCC-KR.

[3] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole, "Buffer OVerflows: Attacks and Defenses for the Vulnerability of the Decade", SANS 2000.

[4] 포항공대 유닉스 보안연구회, "Security PLUS for UNIX", 영진.com

[5] strace man page, Linux man version 1.5j.

제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)